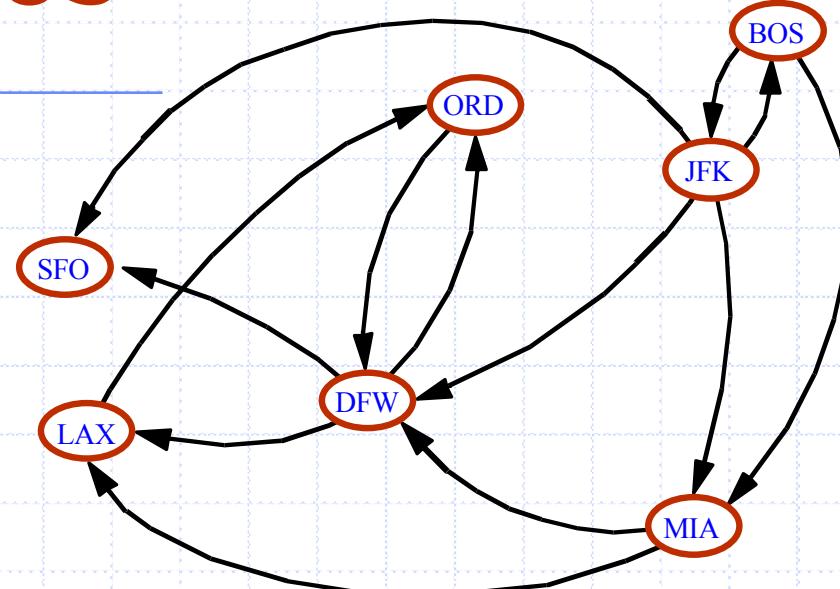


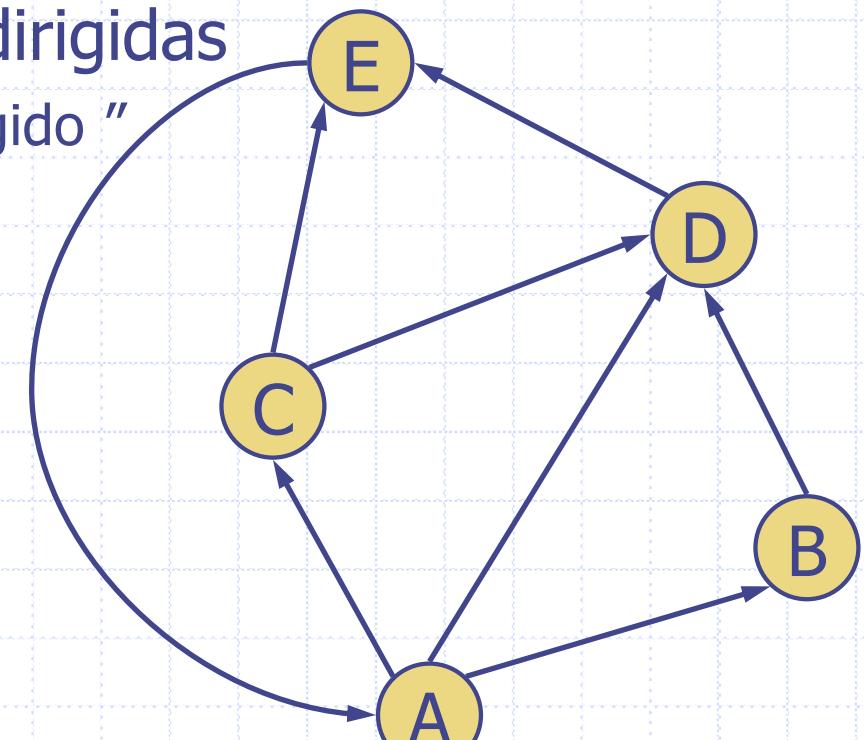
Presentación para usar con el libro de texto **Algorithm Design and Applications**, de MT Goodrich y R. Tamassia, Wiley, 2015

# Grafos Dirigidos



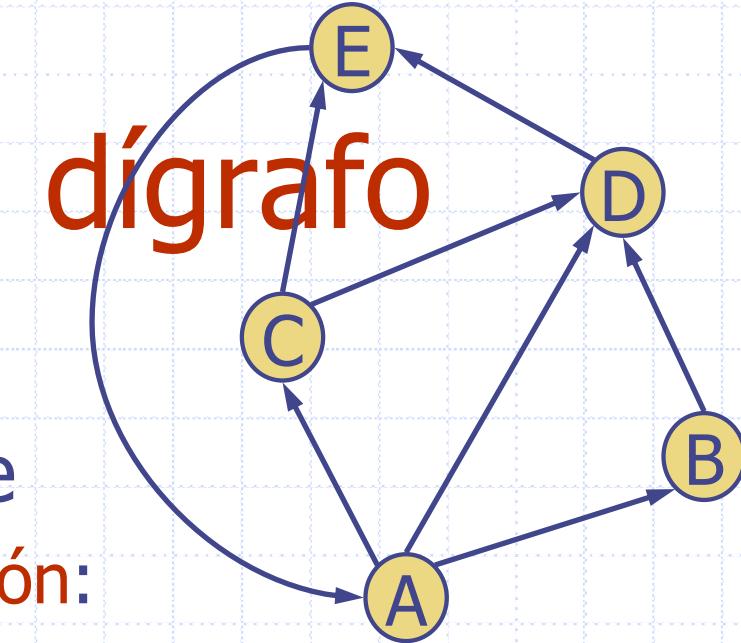
# Grafos Dirigidos (dígrafos)

- Un grafo dirigido es un grafo cuyas aristas están todas dirigidas
  - Abreviatura de " gráfico dirigido "
- Aplicaciones
  - calles de sentido único
  - vuelos
  - programación de tareas



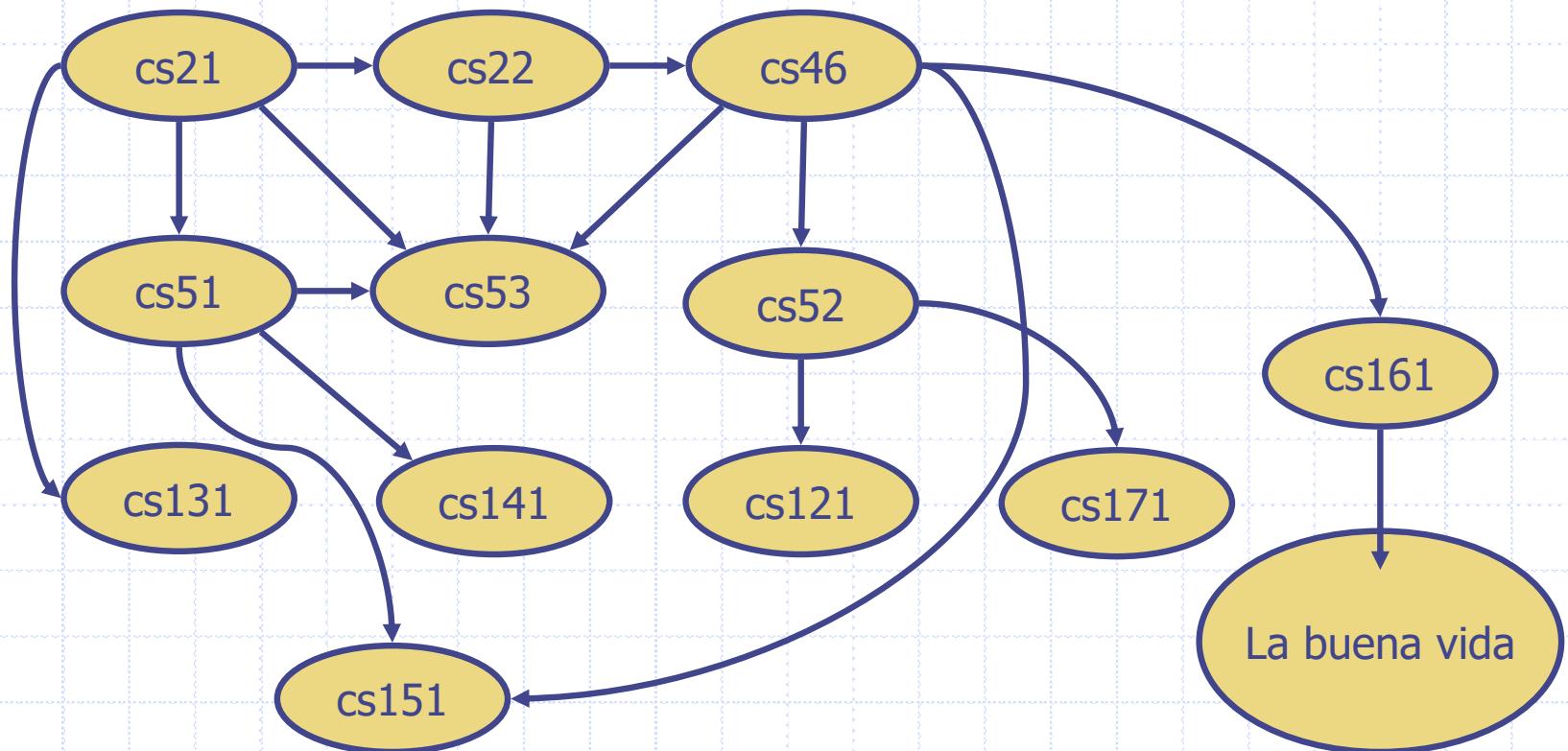
# Propiedades de un dígrafo

- Una gráfica  $G=(V,E)$  tal que
  - Cada arista va en **una dirección**:
  - La arista  $(a, b)$  va de  $a$  a  $b$ , pero no de  $b$  a  $a$
- Si  $G$  es simple,  $m \leq n \cdot (n - 1)$
- Si mantenemos los bordes de entrada y de salida en listas de adyacencia separadas, podemos realizar una lista de las aristas de entrada y de salida en un tiempo proporcional a su tamaño.



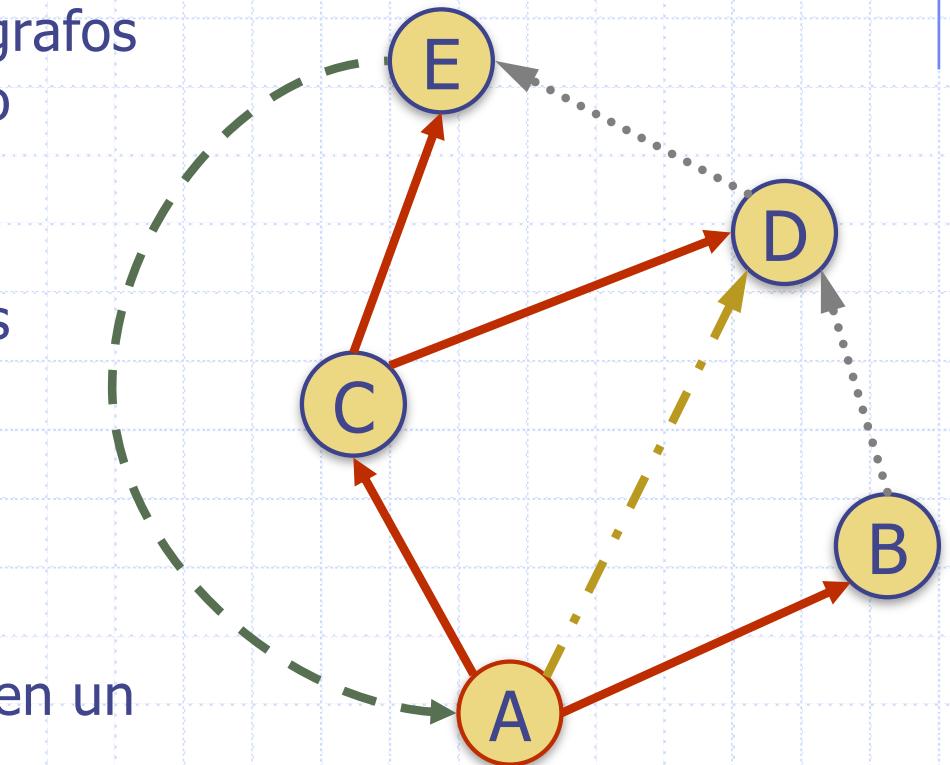
# Aplicación de dígrafo

- **Scheduling:** una arista **(a, b)** significa que la tarea **a** debe completarse antes de que **b** pueda comenzar



# DFS dirigido

- Podemos especializar los algoritmos transversales (DFS y BFS) a dígrafos al atravesar las aristas solo a lo largo de su dirección
- En el algoritmo DFS dirigido, tenemos cuatro tipos de aristas
  - aristas de descubrimiento
  - aristas traseras
  - aristas delanteras
  - aristas cruzadas
- Un DFS dirigido que comienza en un vértice  $s$  determina los vértices **alcanzables desde  $s$**



# El algoritmo DFS dirigido

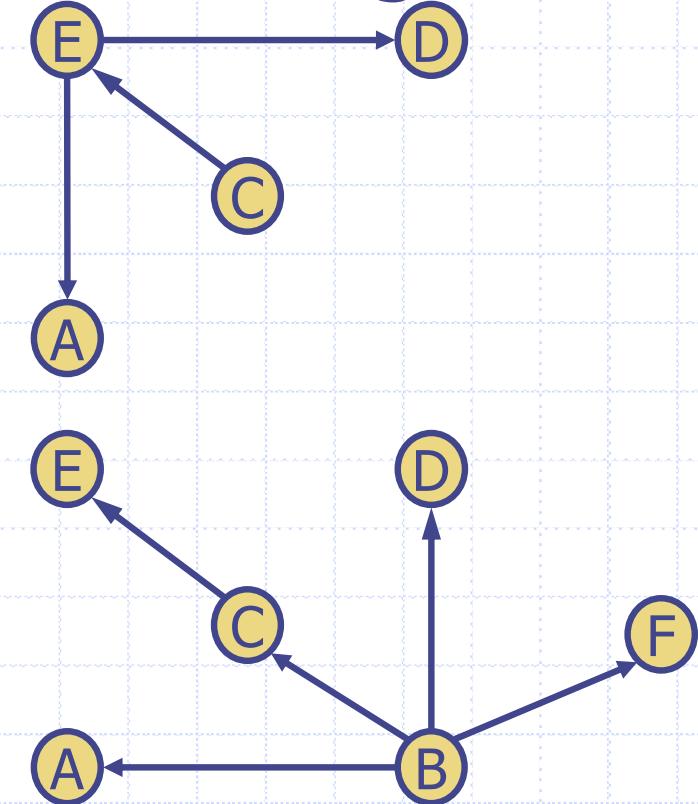
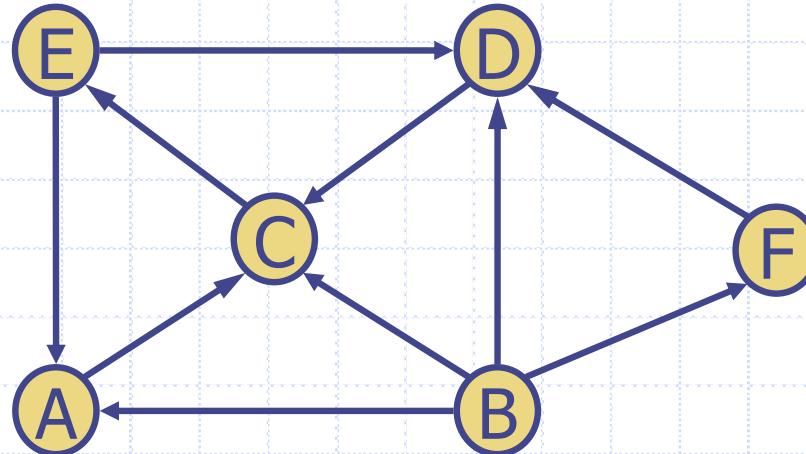
**Algorithm** DirectedDFS( $G, v$ ):

    Label  $v$  as active       // Every vertex is initially unexplored  
    **for** each outgoing edge,  $e$ , that is incident to  $v$  in  $G$  **do**  
        **if**  $e$  is unexplored **then**  
            Let  $w$  be the destination vertex for  $e$   
            **if**  $w$  is unexplored and not active **then**  
                Label  $e$  as a discovery edge  
                DirectedDFS( $G, w$ )  
            **else if**  $w$  is active **then**  
                Label  $e$  as a back edge  
            **else**  
                Label  $e$  as a forward/cross edge  
    Label  $v$  as explored

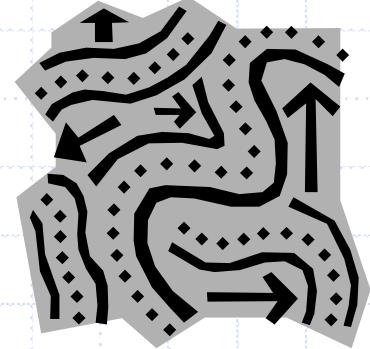
# Accesibilidad



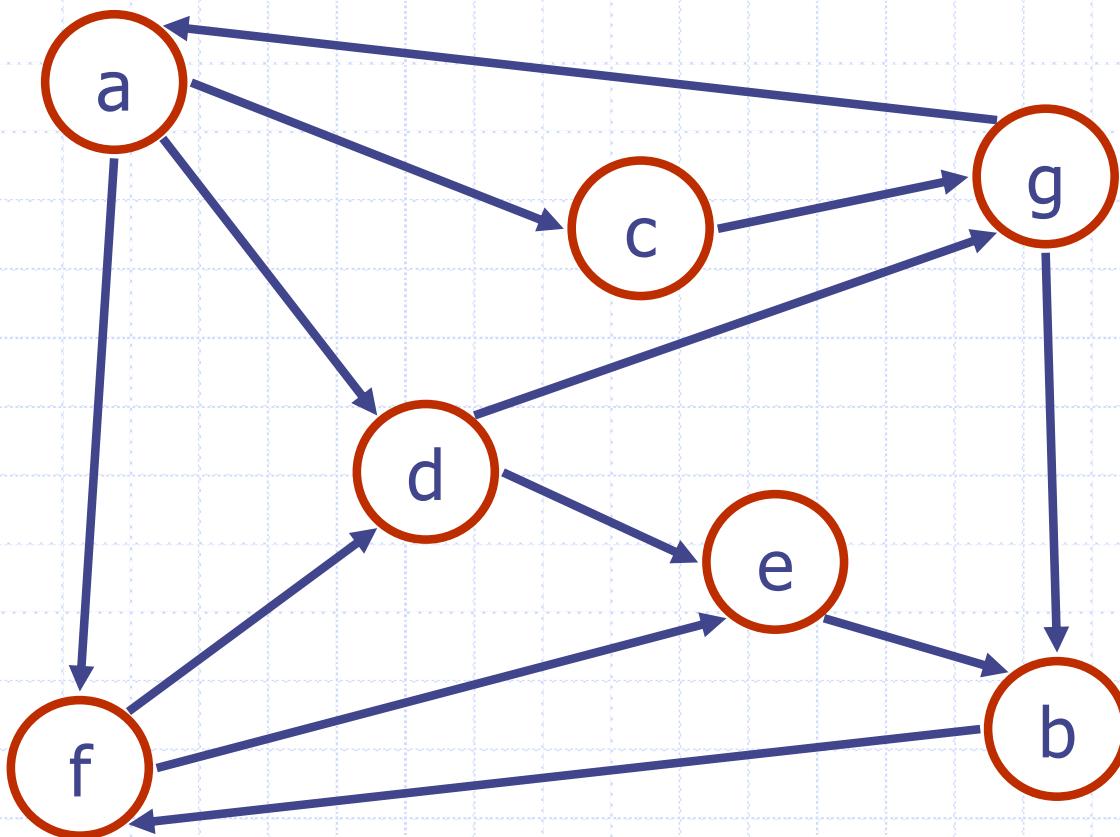
- Árbol DFS con raíz  $v$ : vértices accesibles desde  $v$  a través de caminos dirigidos



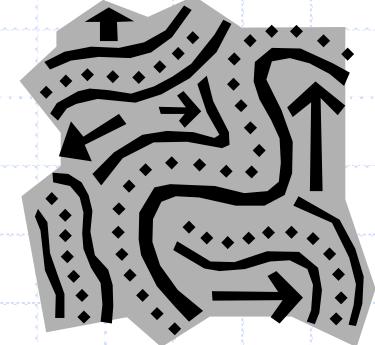
# Conectividad Fuerte



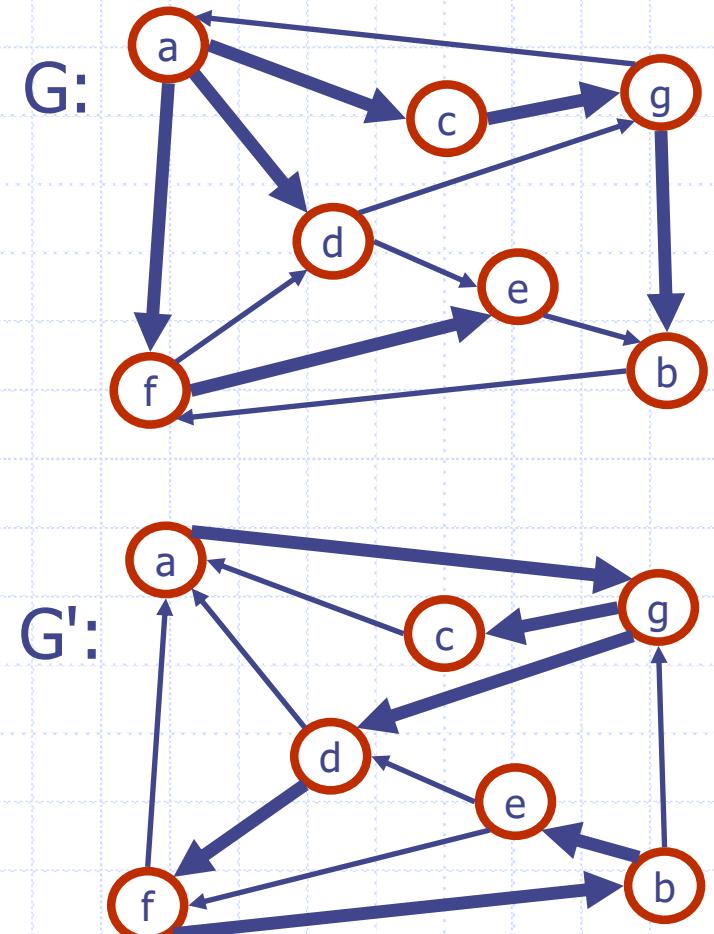
- Cada vértice puede llegar a todos los demás vértices



# Algoritmo de conectividad fuerte



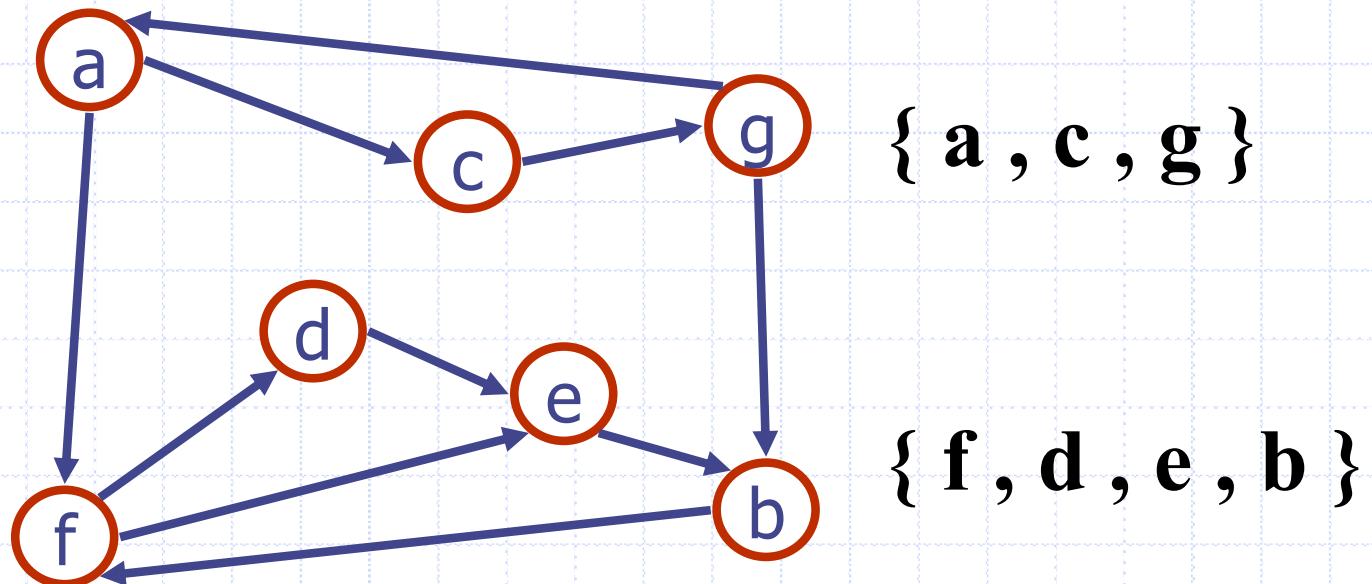
- Elija un vértice **v** en **G**
- Realizar un DFS de **v** en **G**
  - Si hay un vertice **w** no visitado, escriba "no"
- Sea **G'** el grafo **G** con aristas invertidas
- Realizar un DFS de **v** en **G'**
  - Si hay un vertice **w** no visitado, escriba "no"
  - De lo contrario, escriba "sí"
- Tiempo de ejecución:  $O(n+m)$



# Componentes fuertemente conectadas

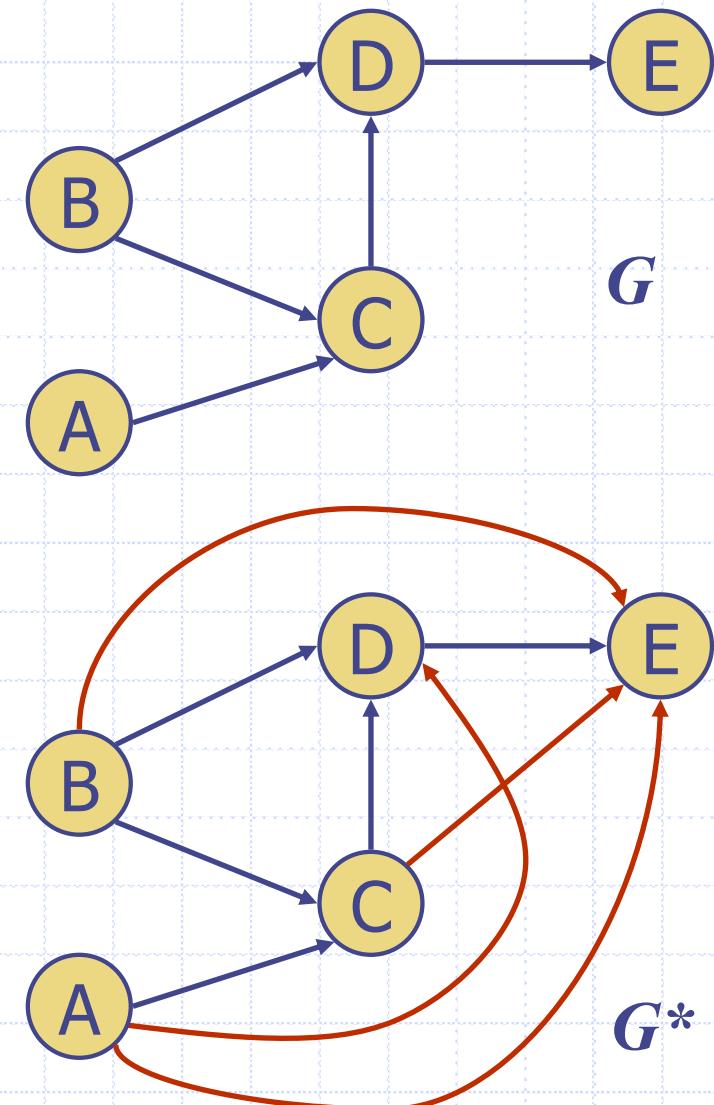


- Subgrafos máximos tales que cada vértice puede alcanzar todos los demás vértices en el subgrafo
- También se puede hacer en tiempo  $O(n+m)$  usando DFS, pero es más complicado (similar a la biconectividad).



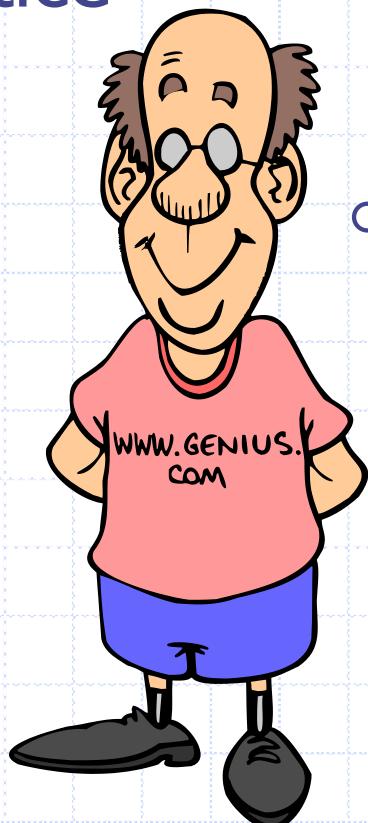
# Clausura transitiva

- Dado un dígrafo  $G$ , la clausura transitiva de  $G$  es el dígrafo  $G^*$  tal que:
  - $G^*$  tiene los mismos vértices que  $G$
  - si  $G$  tiene un camino dirigido de  $u$  a  $v$  ( $u \neq v$ ),  $G^*$  tiene una arista dirigida de  $u$  a  $v$
- El cierre transitivo proporciona información de accesibilidad sobre un dígrafo



# Cálculo del cierre transitivo

- Podemos realizar DFS comenzando en cada vértice
  - $O(n(n+m))$

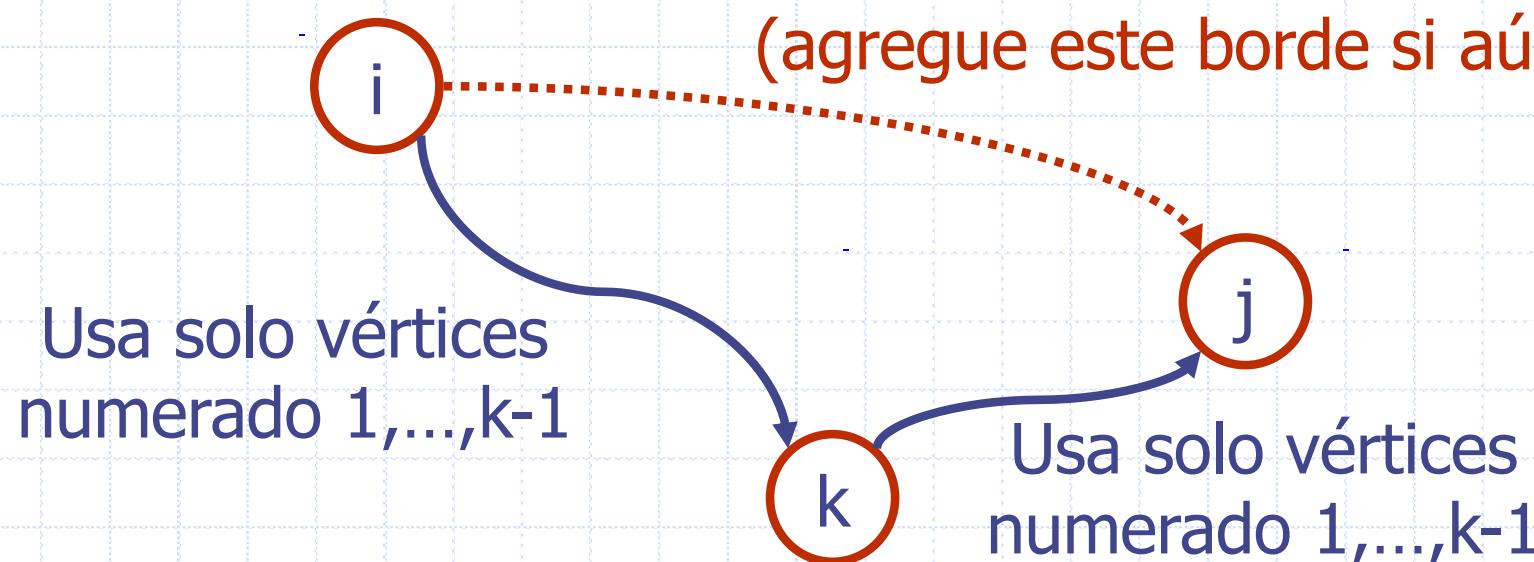


Si hay alguna forma de llegar desde A hasta B y de B hacia C, entonces también habrá alguna de llegar desde A hacia C.

Alternativamente... Usar programación dinámica:  
El Algoritmo de Floyd-Warshall

# Cierre transitivo Floyd-Warshall

- Idea #1: Numerar los vértices 1, 2, ..., n.
- Idea #2: Considere caminos que usan solo vértices numerados 1, 2, ..., k, como vértices intermedios:



# Algoritmo de Floyd-Warshall: Vista de alto nivel



- Número de vértices  $v_1, \dots, v_n$
- Calcular dígrafos  $G_0, \dots, G_n$ 
  - $G_0 = G$
  - $G_k$  tiene una arista dirigida  $(v_i, v_j)$  si  $G$  tiene un camino dirigido desde  $v_i$  hasta  $v_j$  con vértices intermedios en  $\{v_1, \dots, v_k\}$
- Tenemos que  $G_n = G^*$
- En la fase  $k$ , el dígrafo  $G_k$  se calcula a partir de  $G_{k-1}$
- Tiempo de ejecución:  $O(n^3)$ , suponiendo que **sonAdyacentes()** es de  $O(1)$  (e.g., matriz de adyacencia)

# El algoritmo de Floyd-Warshall

**Algorithm FloydWarshall( $\vec{G}$ ):**

**Input:** A digraph  $\vec{G}$  with  $n$  vertices

**Output:** The transitive closure  $\vec{G}^*$  of  $\vec{G}$

Let  $v_1, v_2, \dots, v_n$  be an arbitrary numbering of the vertices of  $\vec{G}$

$\vec{G}_0 \leftarrow \vec{G}$

**for**  $k \leftarrow 1$  **to**  $n$  **do**

$\vec{G}_k \leftarrow \vec{G}_{k-1}$

**for**  $i \leftarrow 1$  **to**  $n, i \neq k$  **do**

**for**  $j \leftarrow 1$  **to**  $n, j \neq i, k$  **do**

**if** both edges  $(v_i, v_k)$  and  $(v_k, v_j)$  are in  $\vec{G}_{k-1}$  **then**

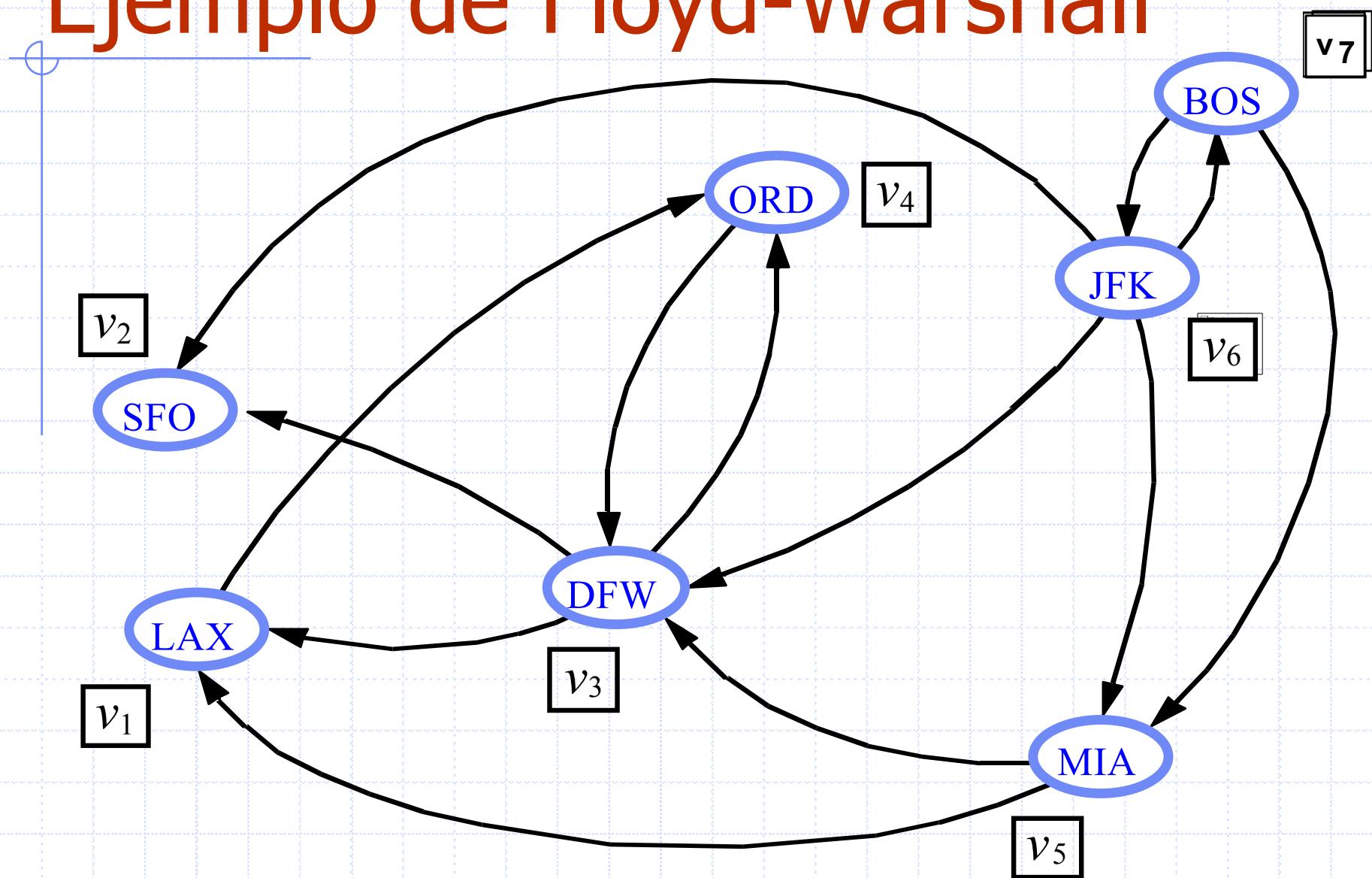
**if**  $\vec{G}_k$  does not contain directed edge  $(v_i, v_j)$  **then**

                    add directed edge  $(v_i, v_j)$  to  $\vec{G}_k$

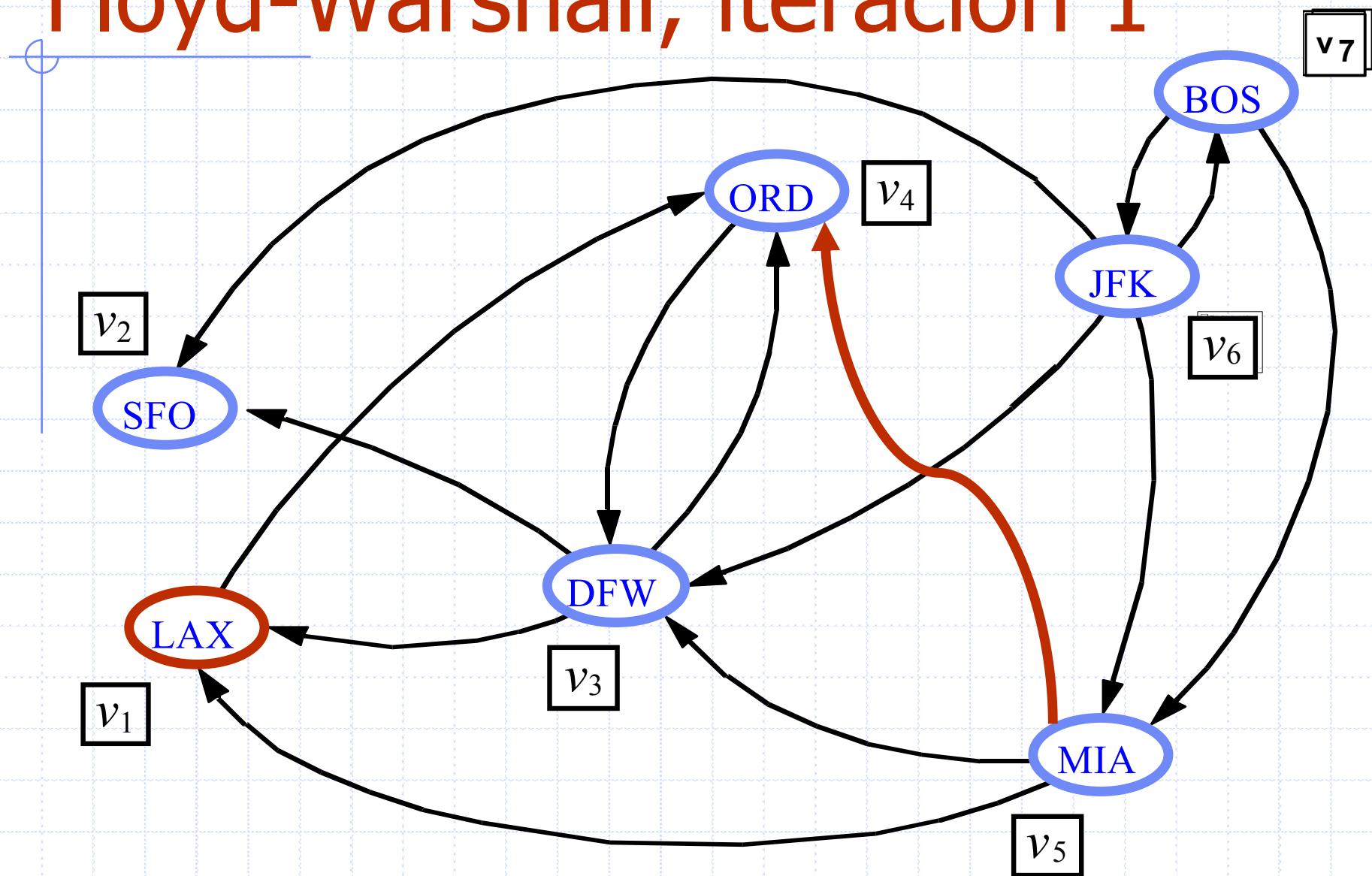
**return**  $\vec{G}_n$

- El tiempo de ejecución es claramente  $O(n^3)$ .

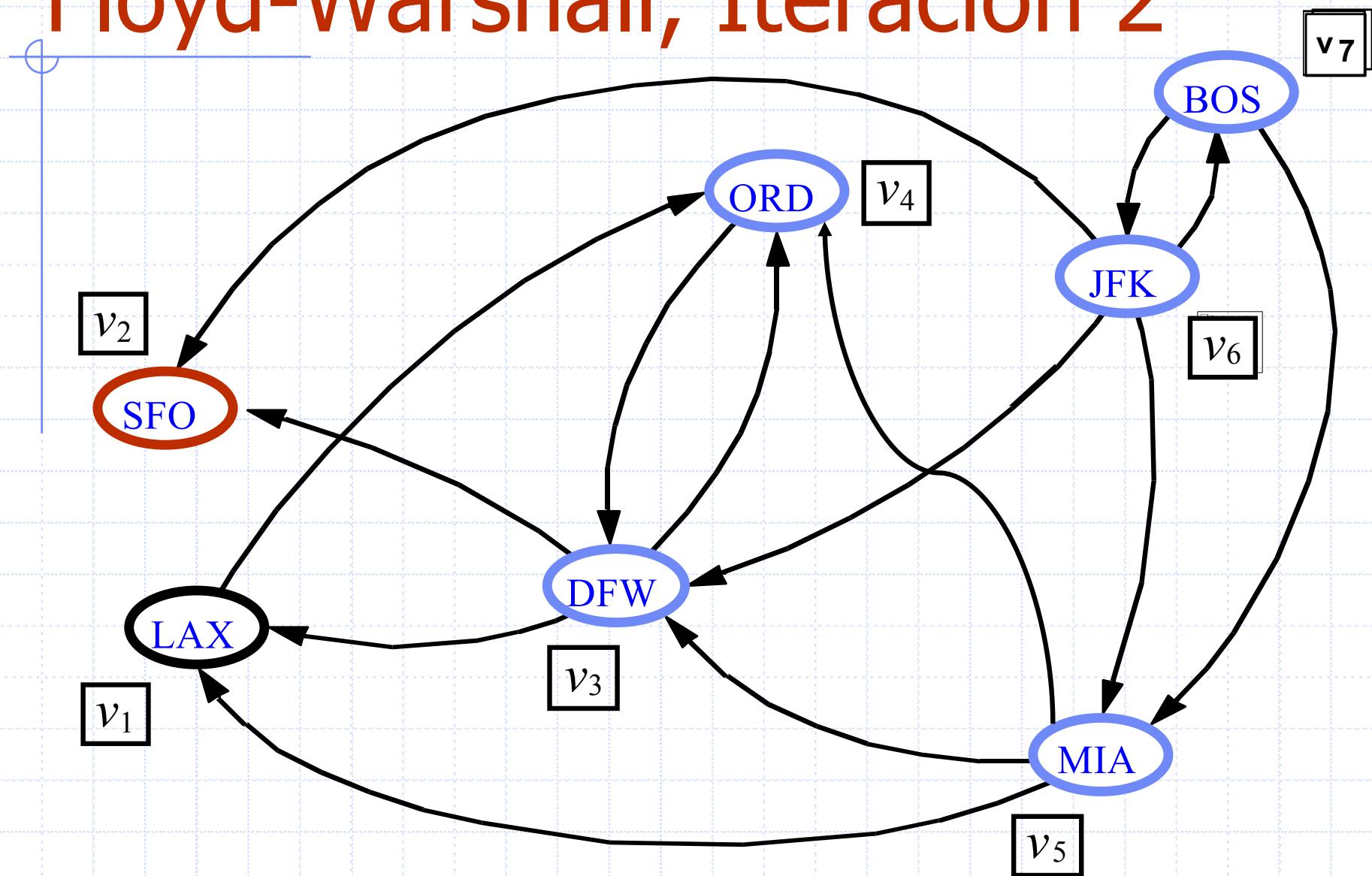
# Ejemplo de Floyd-Warshall



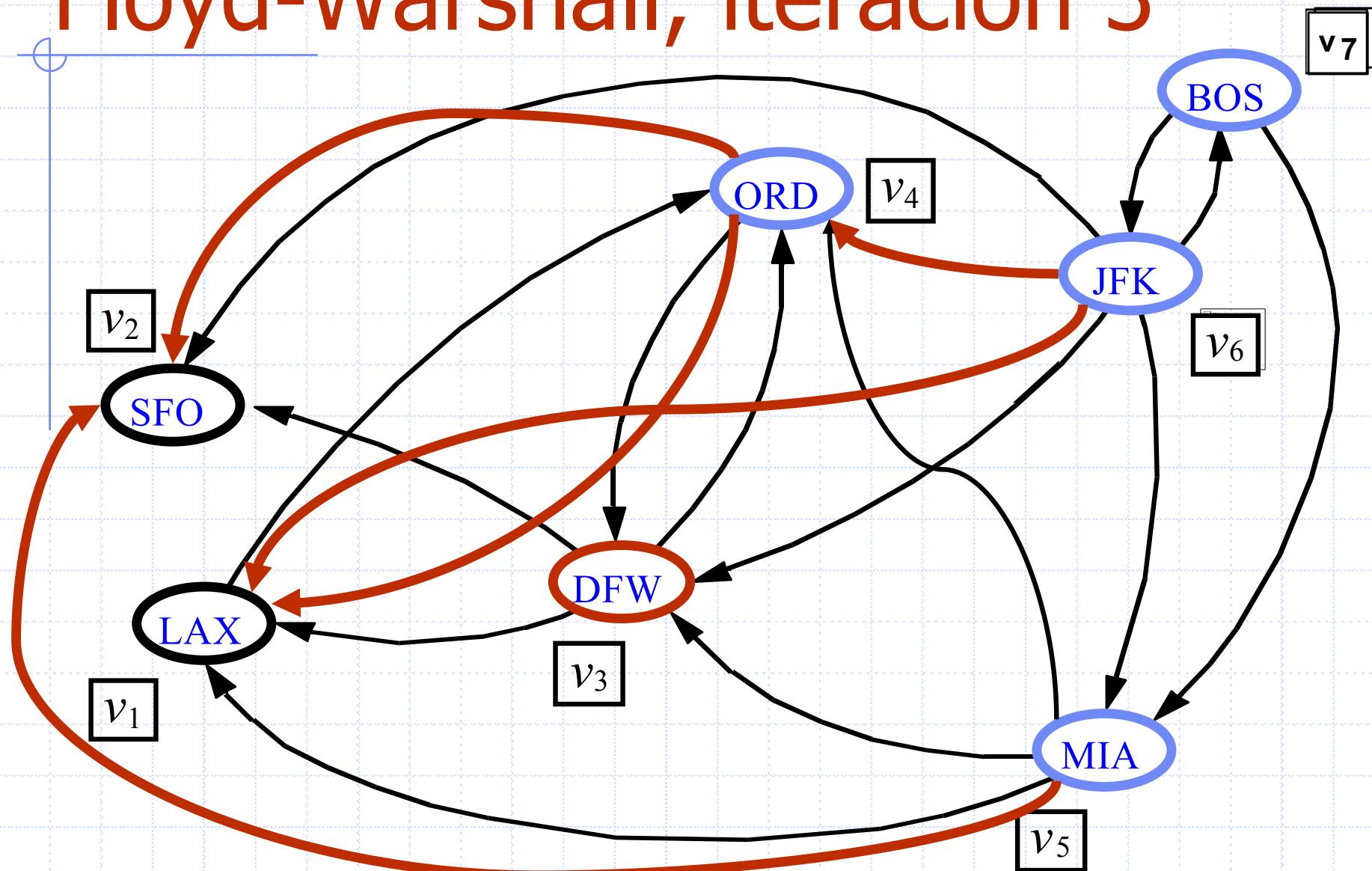
# Floyd-Warshall, iteración 1



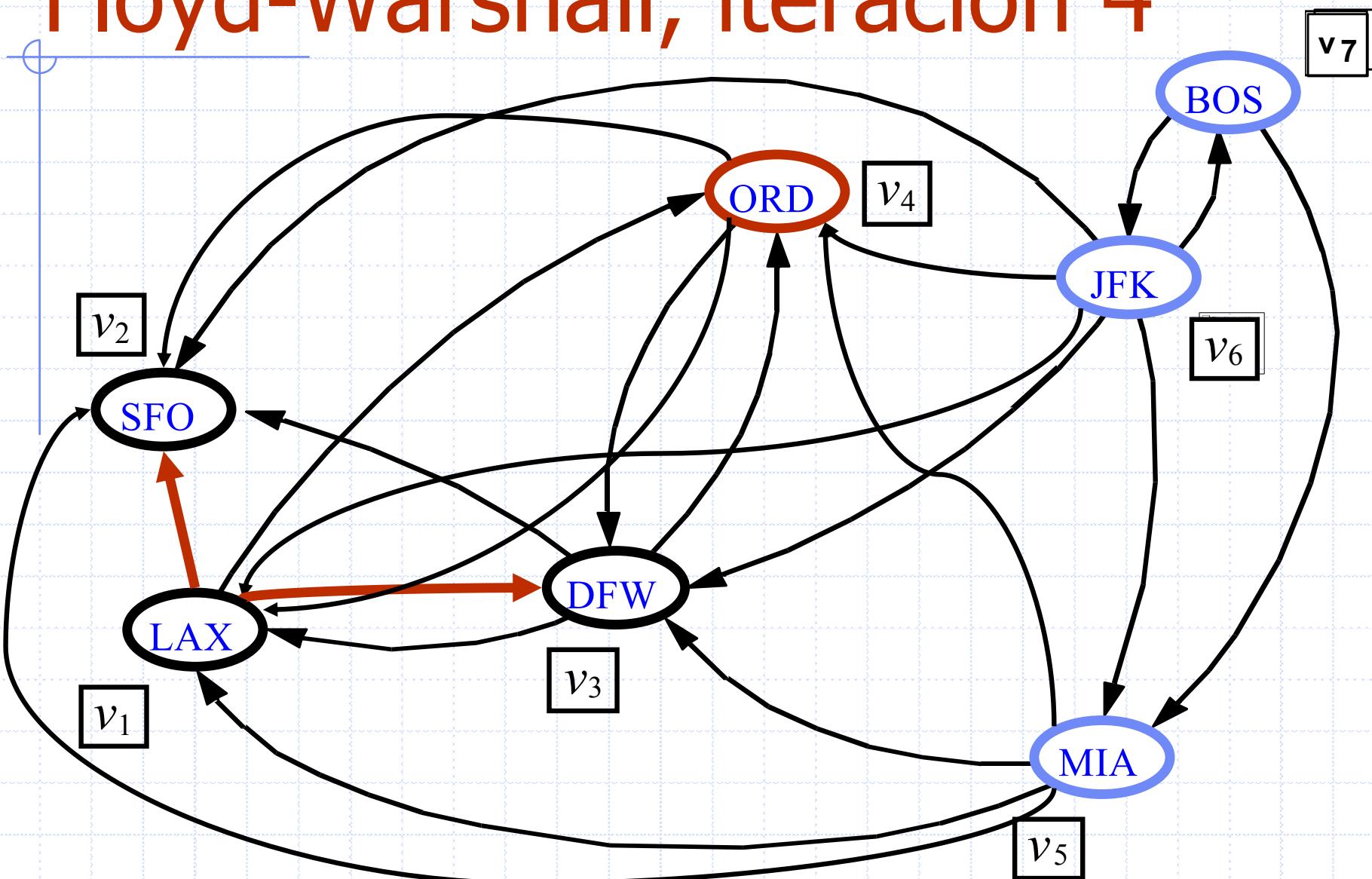
# Floyd-Warshall, Iteración 2



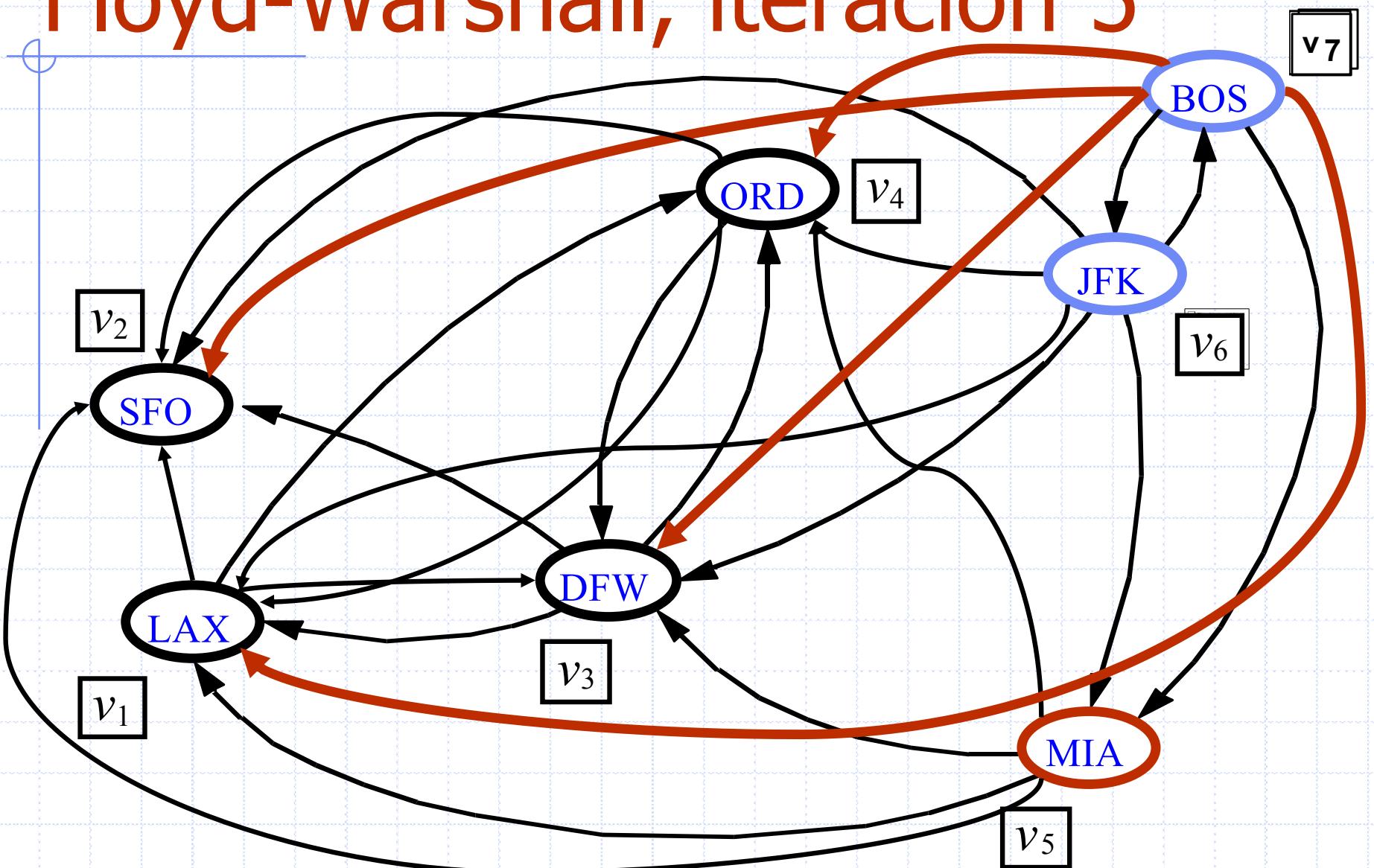
# Floyd-Warshall, iteración 3



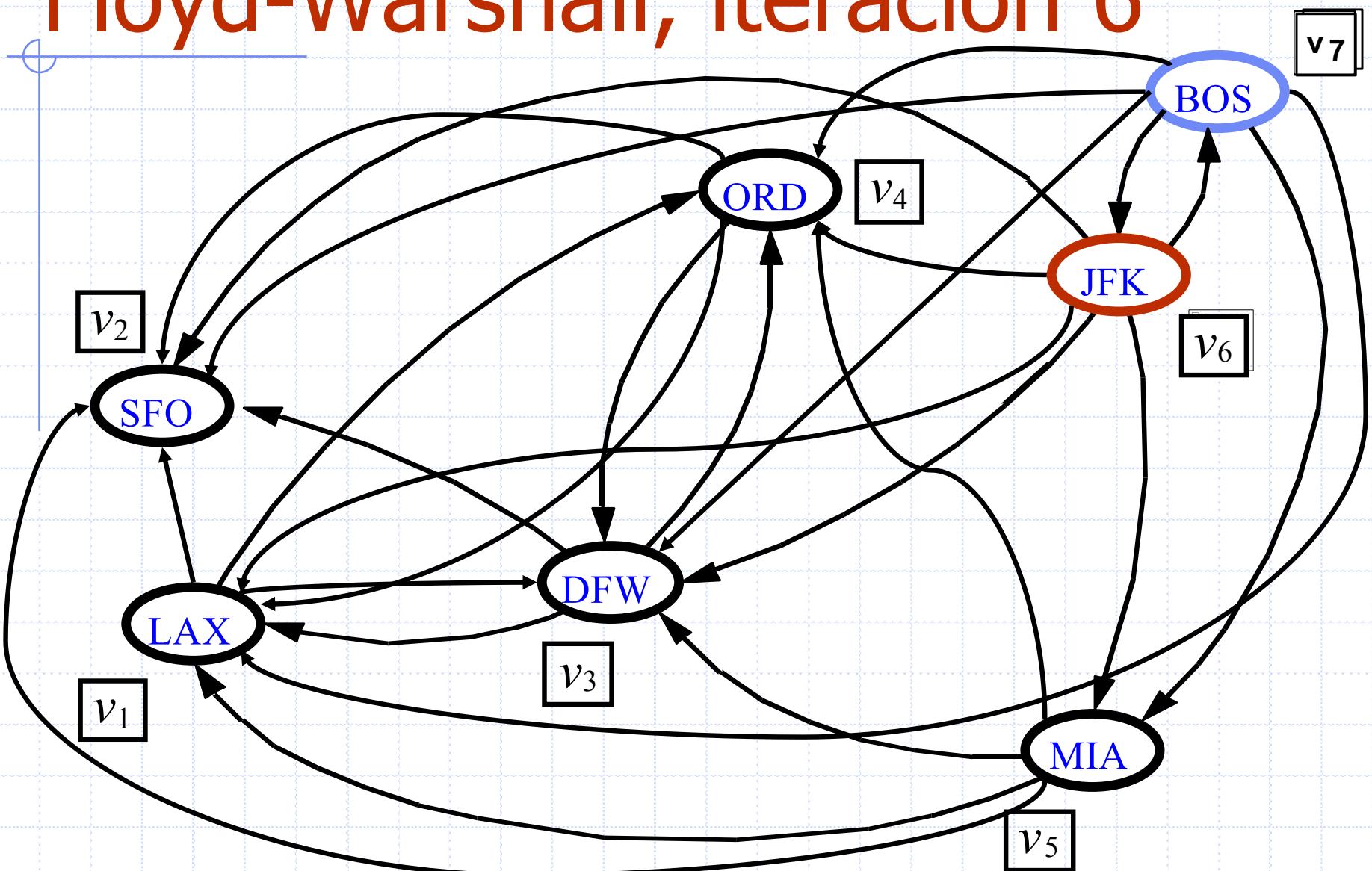
# Floyd-Warshall, iteración 4



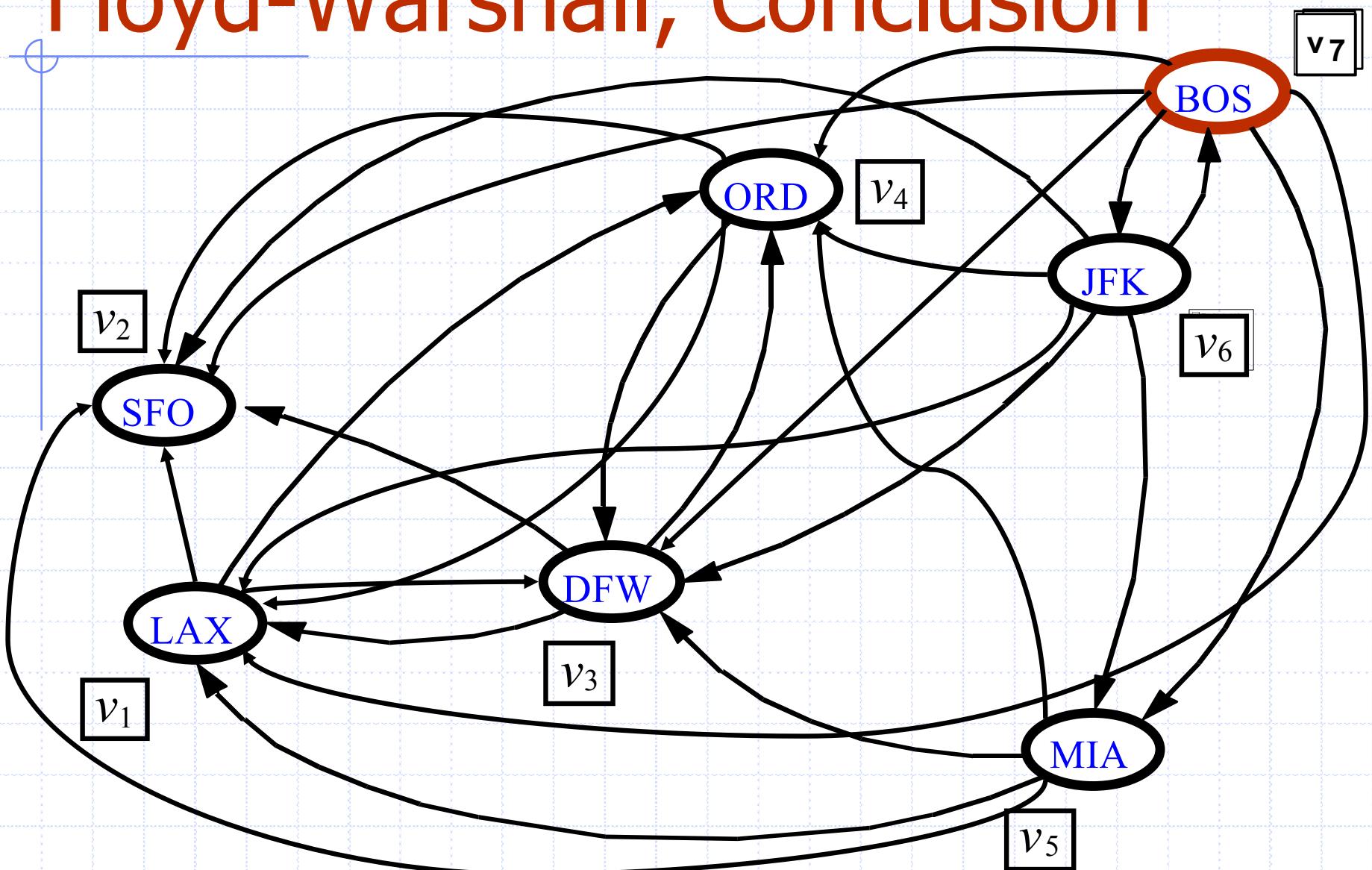
# Floyd-Warshall, iteración 5



# Floyd-Warshall, iteración 6



# Floyd-Warshall, Conclusión



# DAG y ordenación topológica

- Un grafo acíclico dirigido (DAG) es un dígrafo que no tiene ciclos dirigidos
- Una ordenación topológica de un dígrafo es una numeración

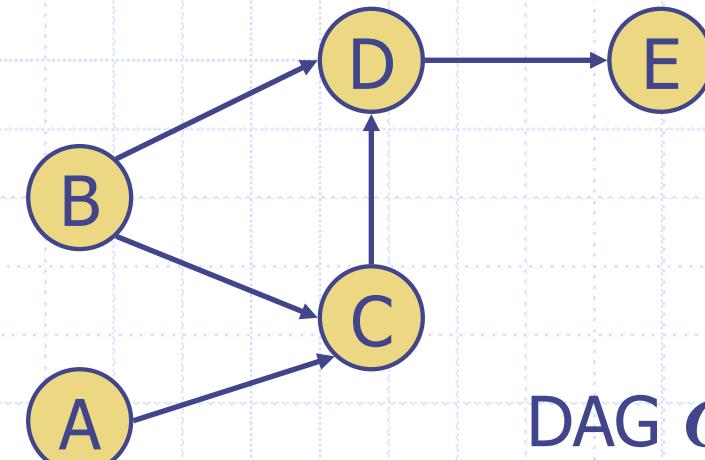
$v_1, \dots, v_n$

de los vértices tal que para cada arista  $(v_i, v_j)$ , tenemos  $i < j$

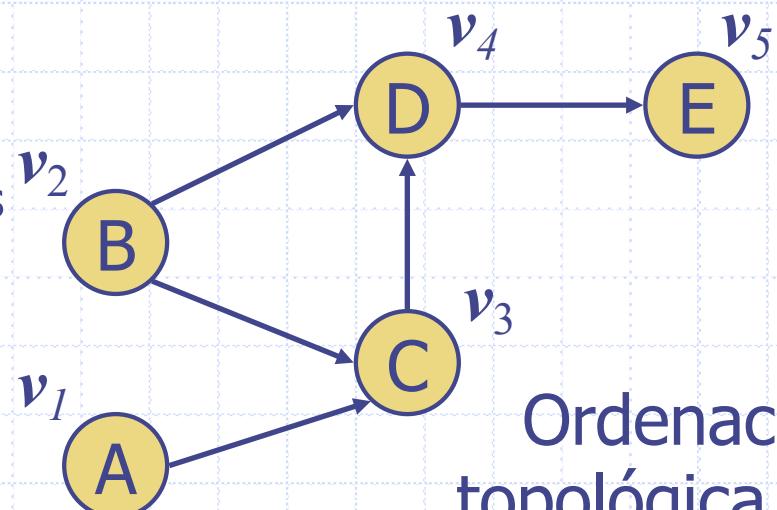
- Ejemplo: en un dígrafo de programación de tareas, una ordenación topológica de una secuencia de tareas que satisface las restricciones de precedencia

## Teorema

Un dígrafo admite una ordenación topológica si y solo si es un DAG

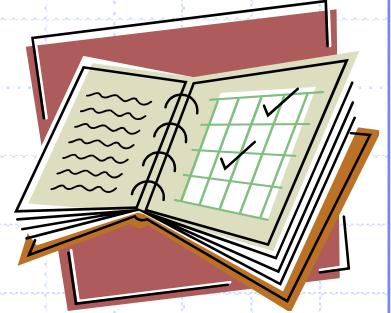


DAG  $G$

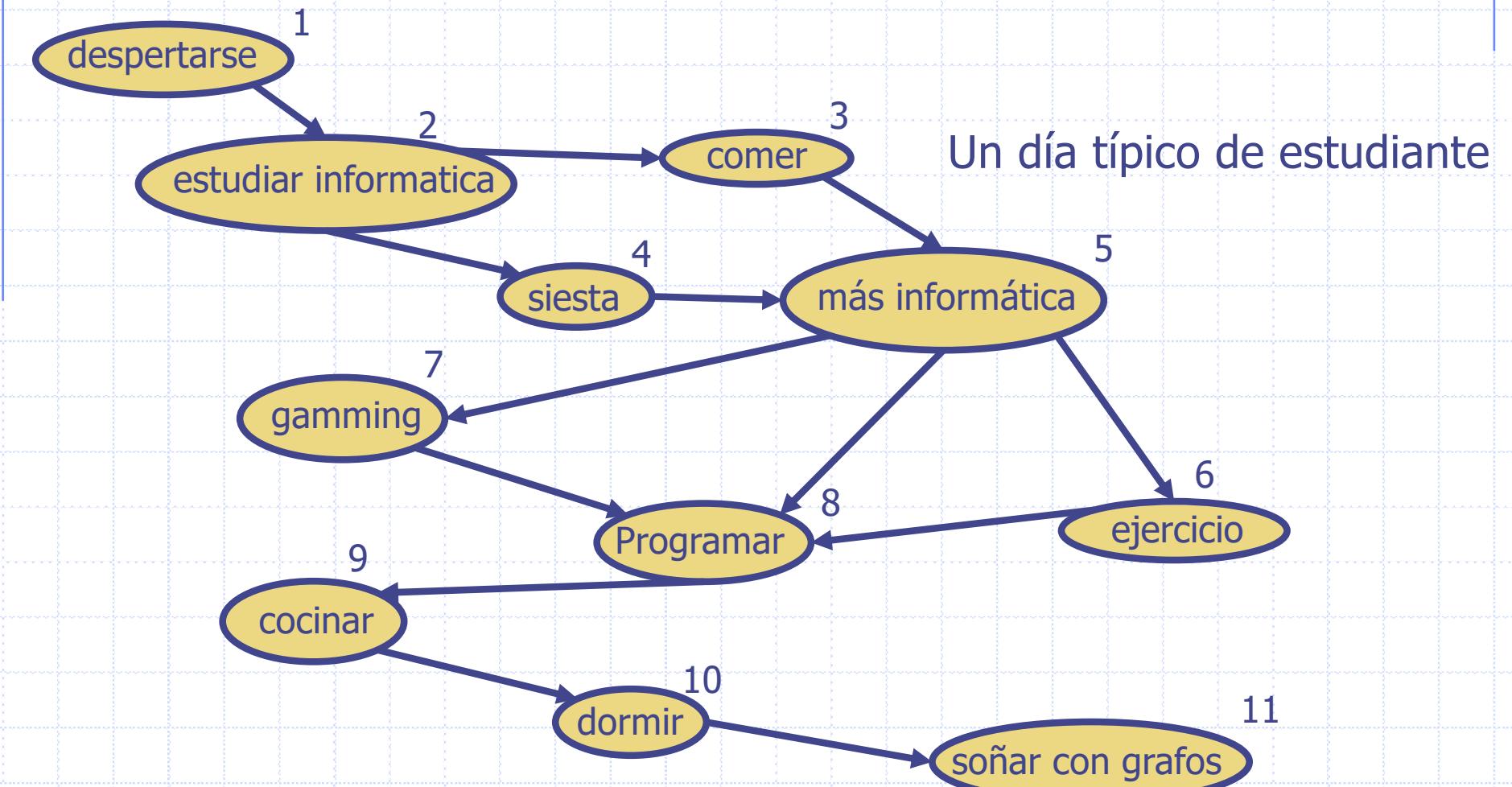


Ordenación topológica de  $G$

# Clasificación Topológica



- Numeramos cada vértice, de modo que  $(u,v)$  en  $E$  implica  $u < v$



# Algoritmo para la Clasificación Topológica

- Nota: Este algoritmo es diferente al del libro.

**Algoritmo** `clasificacionTopologica( $G$ )`

$H \leftarrow G$  #Copia temporal de  $G$

$n \leftarrow G.\text{numVertices}()$

mientras  $H$  no está vacío **hacer:**

    Sea  $v$  un vértice sin aristas salientes

    Etiquetar  $v \leftarrow n$

$n \leftarrow n - 1$

    Eliminar  $v$  de  $H$

- Tiempo de ejecución:  $O(n + m)$

# Implementación con DFS

- Simule el algoritmo utilizando búsqueda en profundidad
- $O(n+m)$  tiempo.

## Algoritmo *DFStopológico(G)*

**Entrada:** un grafo dirigido  $G$

**Salida:** ordenación topológica de  $G$

$n \leftarrow G.\text{numVertices}()$

para todo  $u \in G.\text{vertices}()$

*setLabel* ( $u$ , SIN EXPLORAR )

para todo  $v \in G.\text{vertices}()$

    si *getLabel*( $v$ ) = SIN EXPLORAR

*DFStopológico*( $G$ ,  $v$ )

## Algoritmo *DFStopológico (G, v)*

**Entrada:** un grafo  $G$  y un vértice inicial  $v$  de  $G$

**Salida:** etiquetado de los vértices de  $G$   
en la componente conexa de  $v$

*setLabel* ( $v$ , VISITADO )

para todo  $e \in G.\text{aristasSalientes}(v)$

    { aristas salientes }

$w \leftarrow \text{opuesto}(v, e)$

    if *getLabel* ( $w$ ) = SIN EXPLORAR

        {  $e$  es una arista de descubrimiento }

*DFStopológico*( $G$ ,  $w$ )

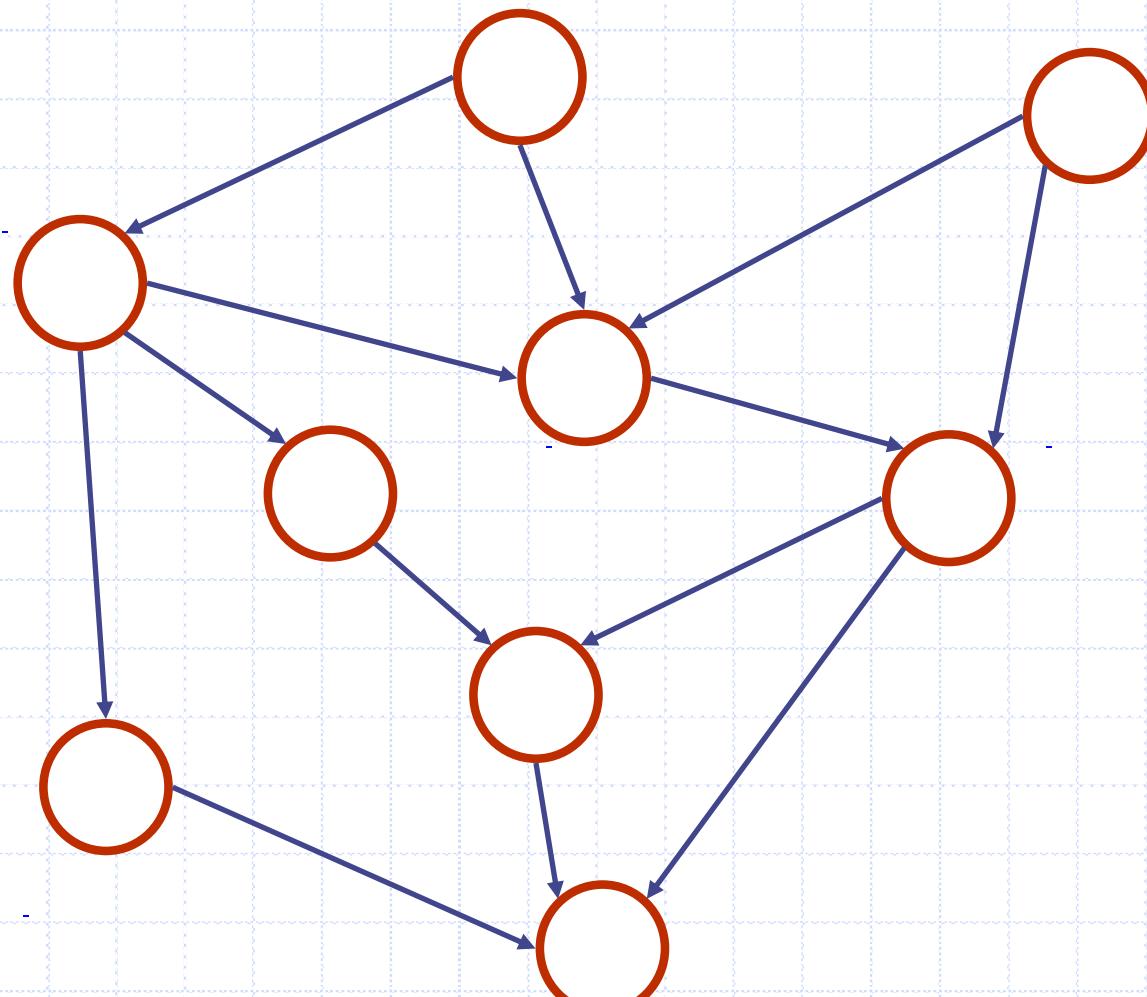
    else

        {  $e$  es una arista saliente o cruzada }

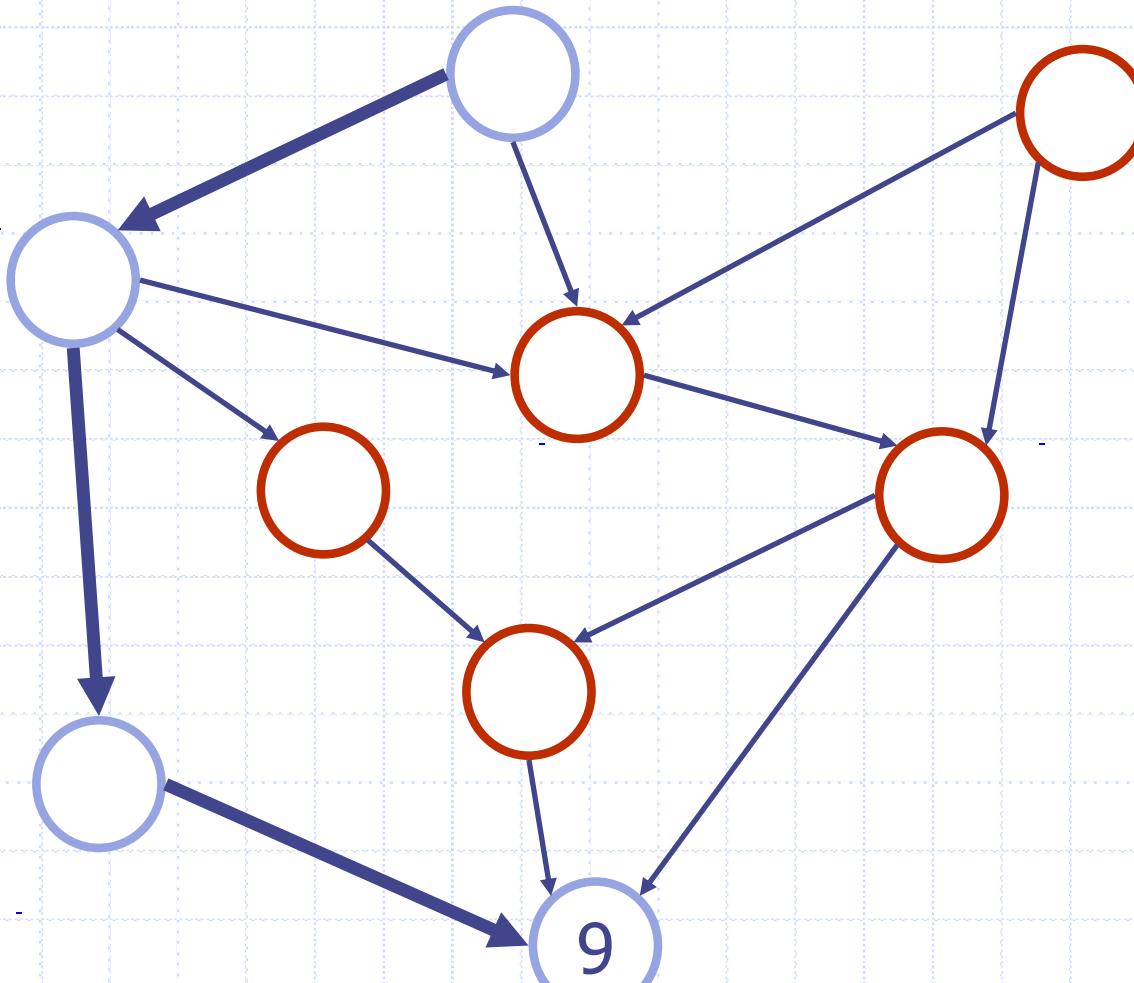
Etiquetar  $v$  con número topológico  $n$

$n \leftarrow n - 1$

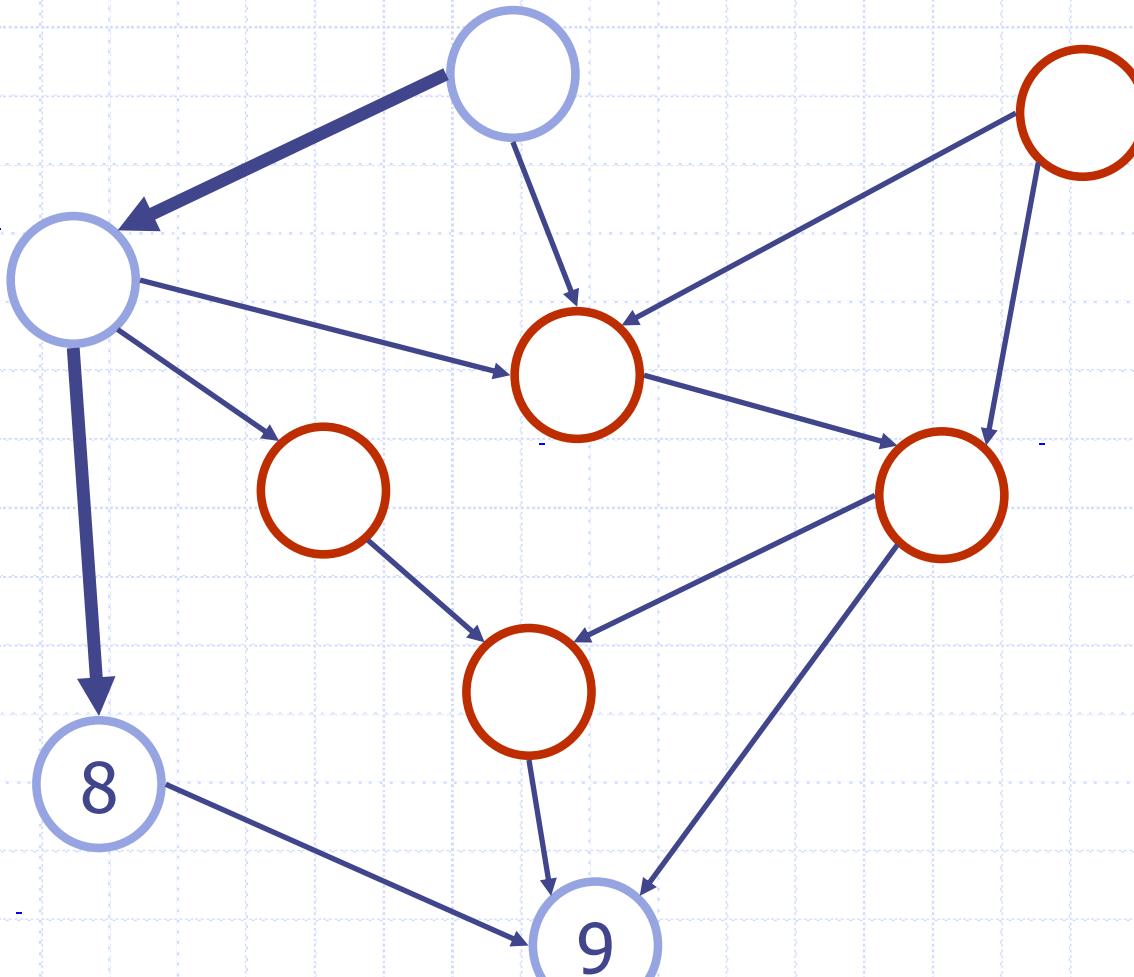
# Ejemplo: Clasificación Topológica



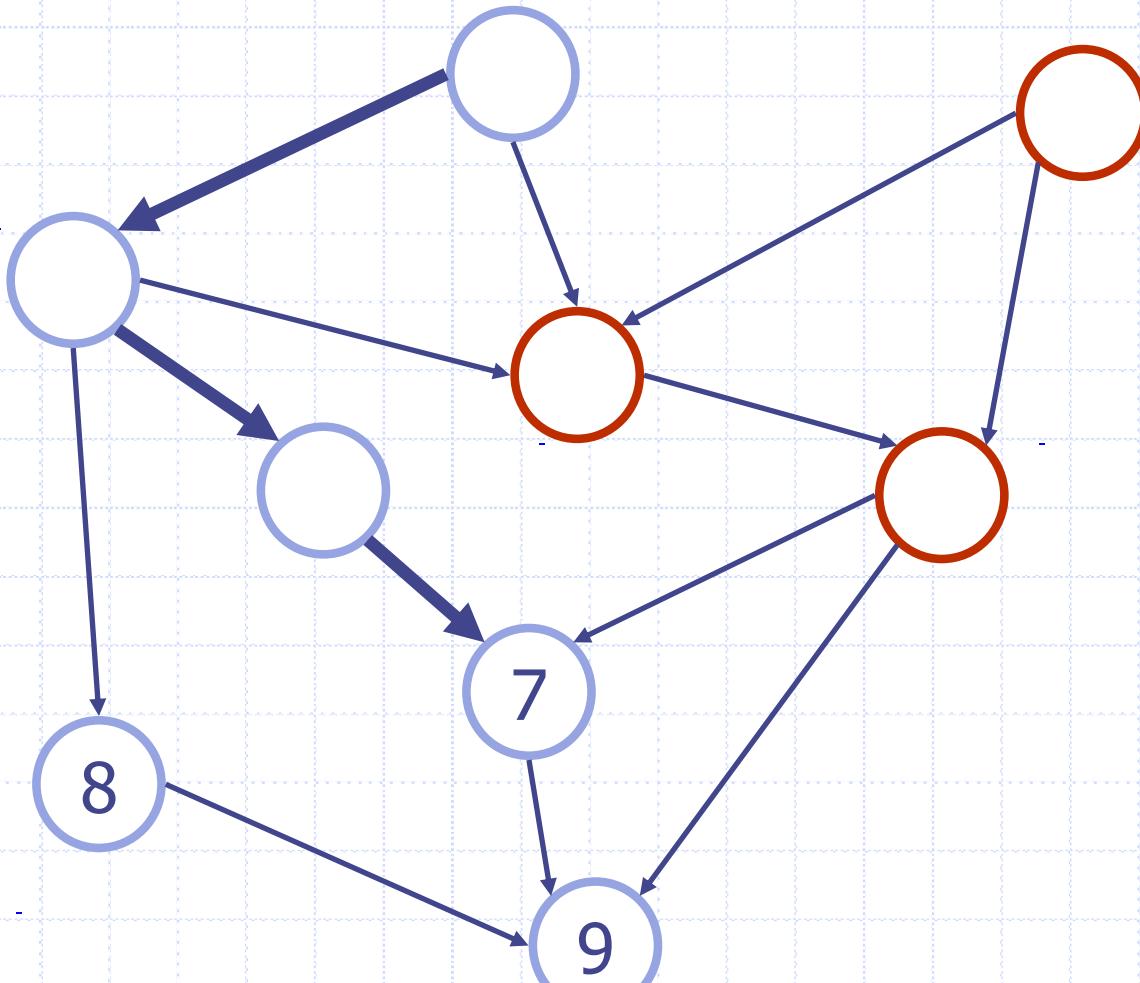
# Ejemplo: Clasificación Topológica



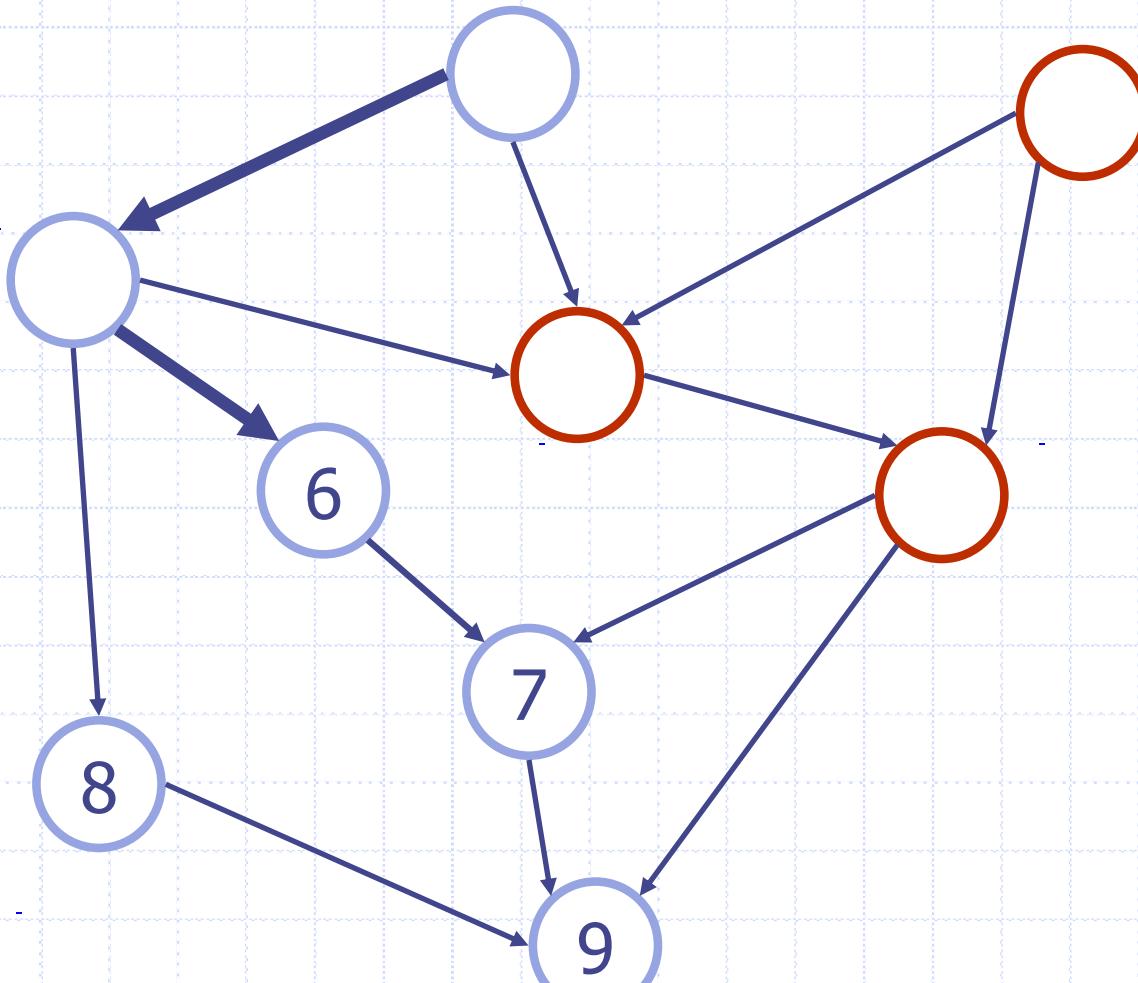
# Ejemplo: Clasificación Topológica



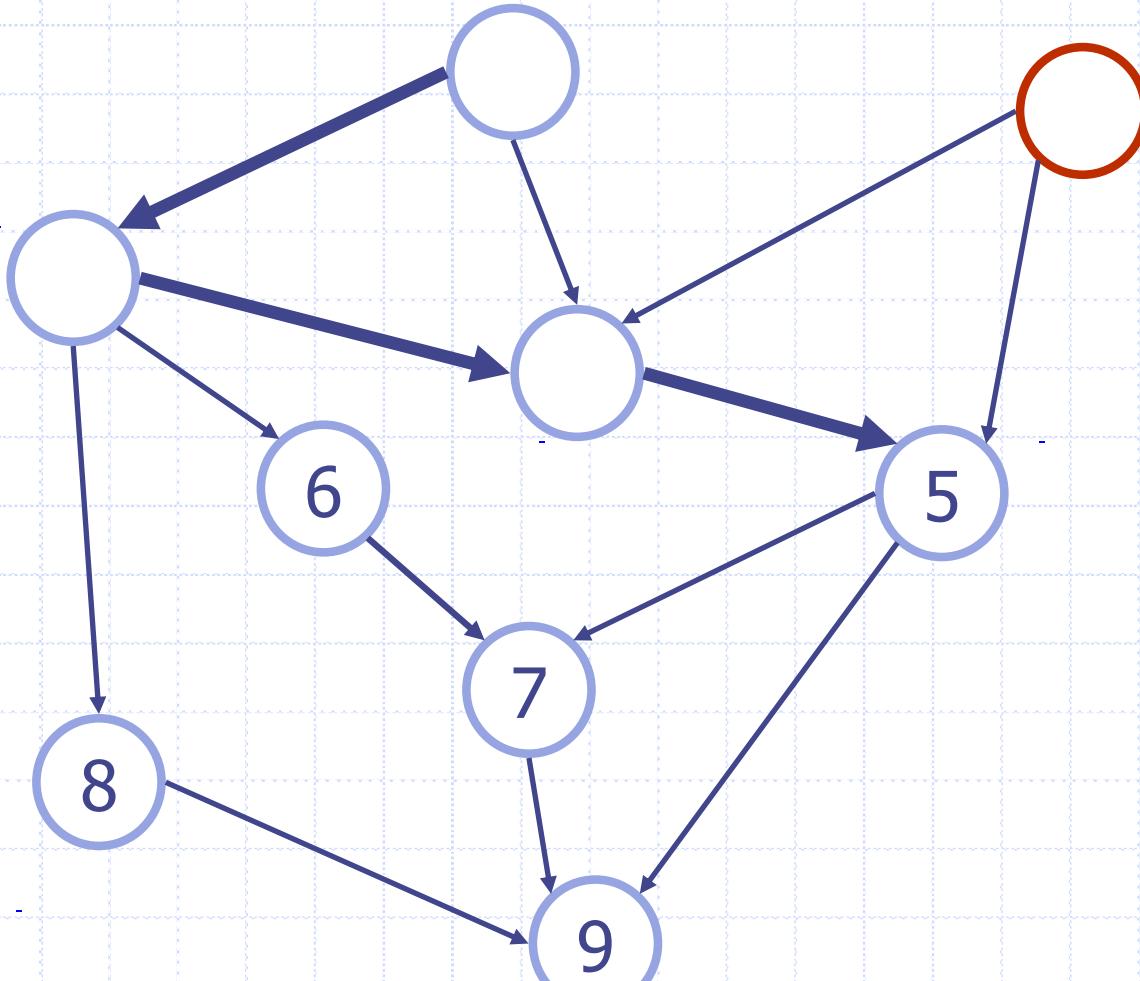
# Ejemplo: Clasificación Topológica



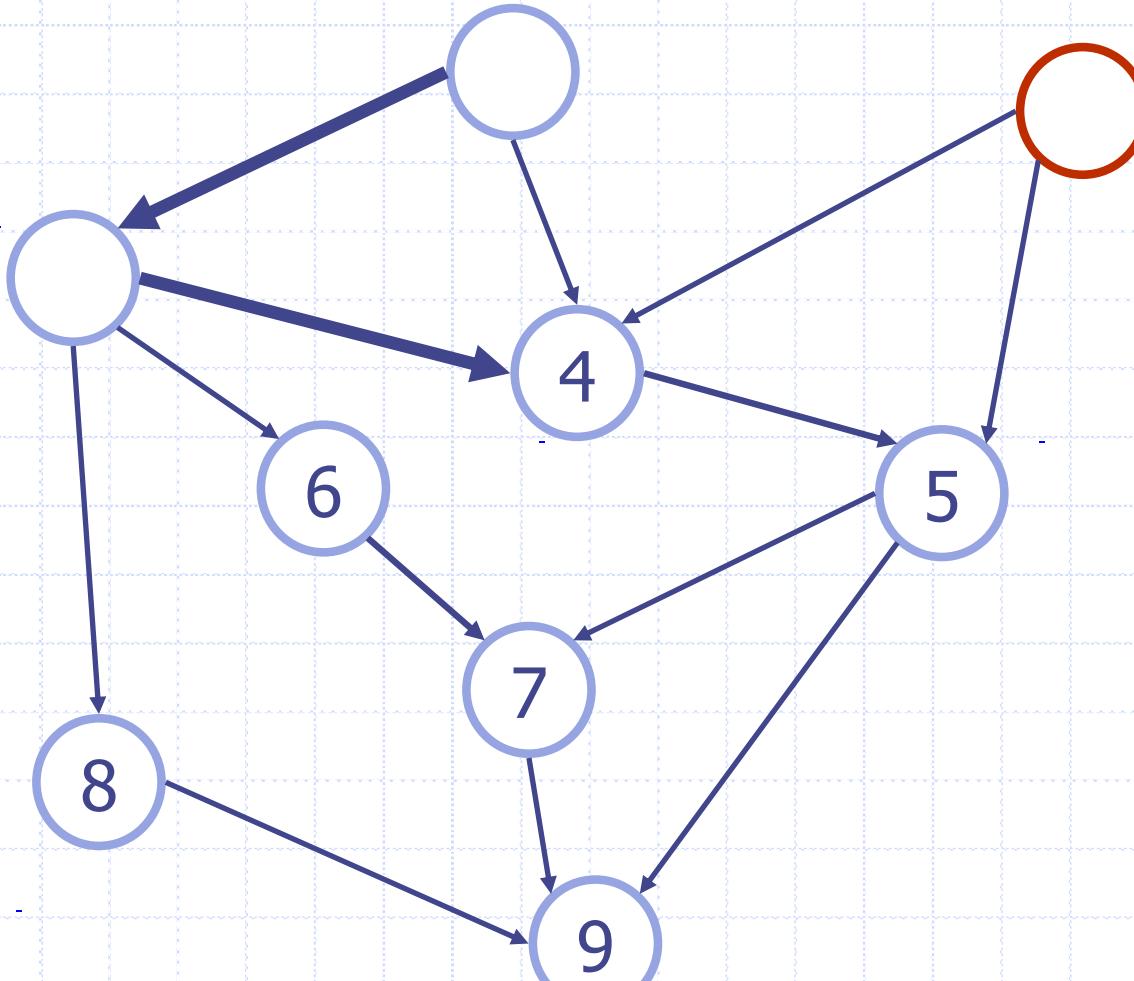
# Ejemplo: Clasificación Topológica



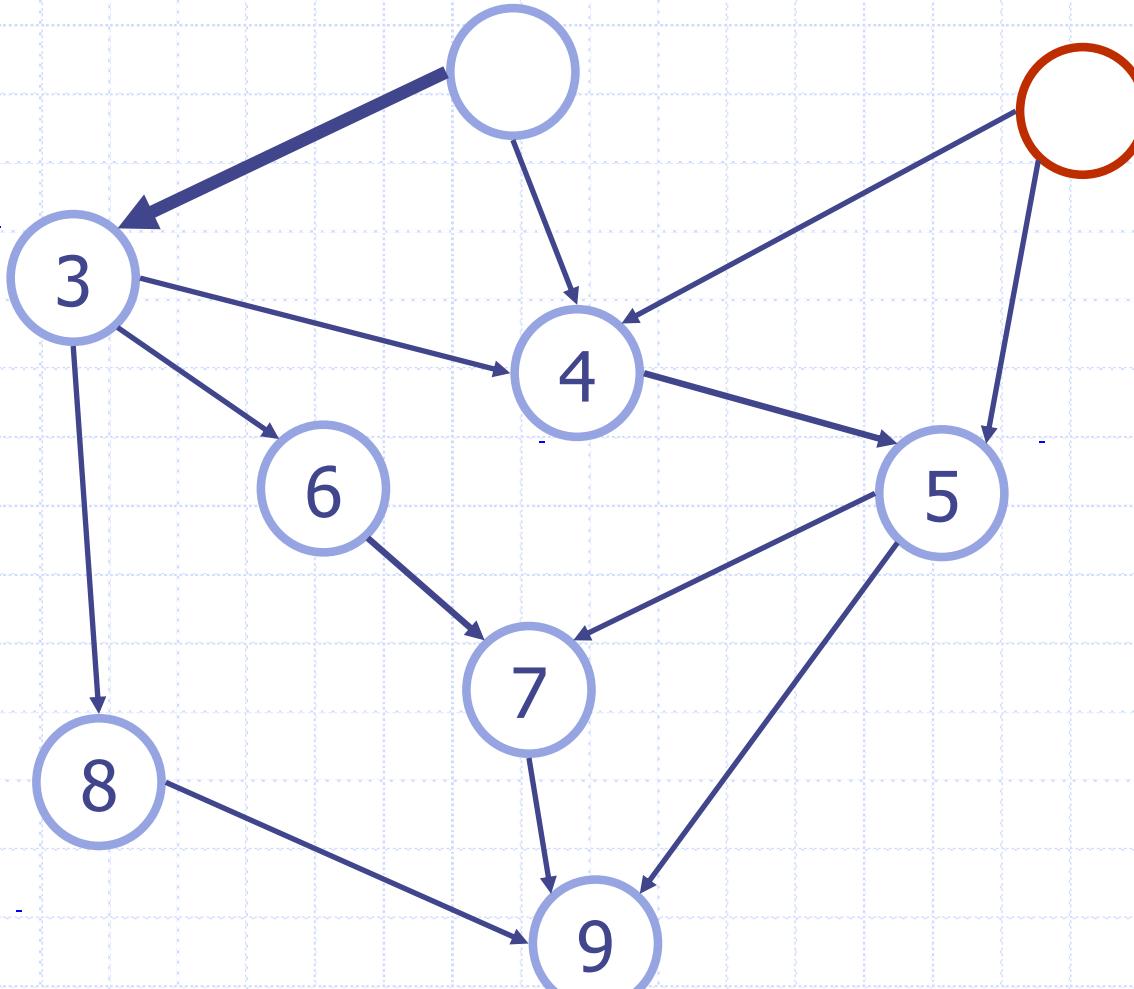
# Ejemplo: Clasificación Topológica



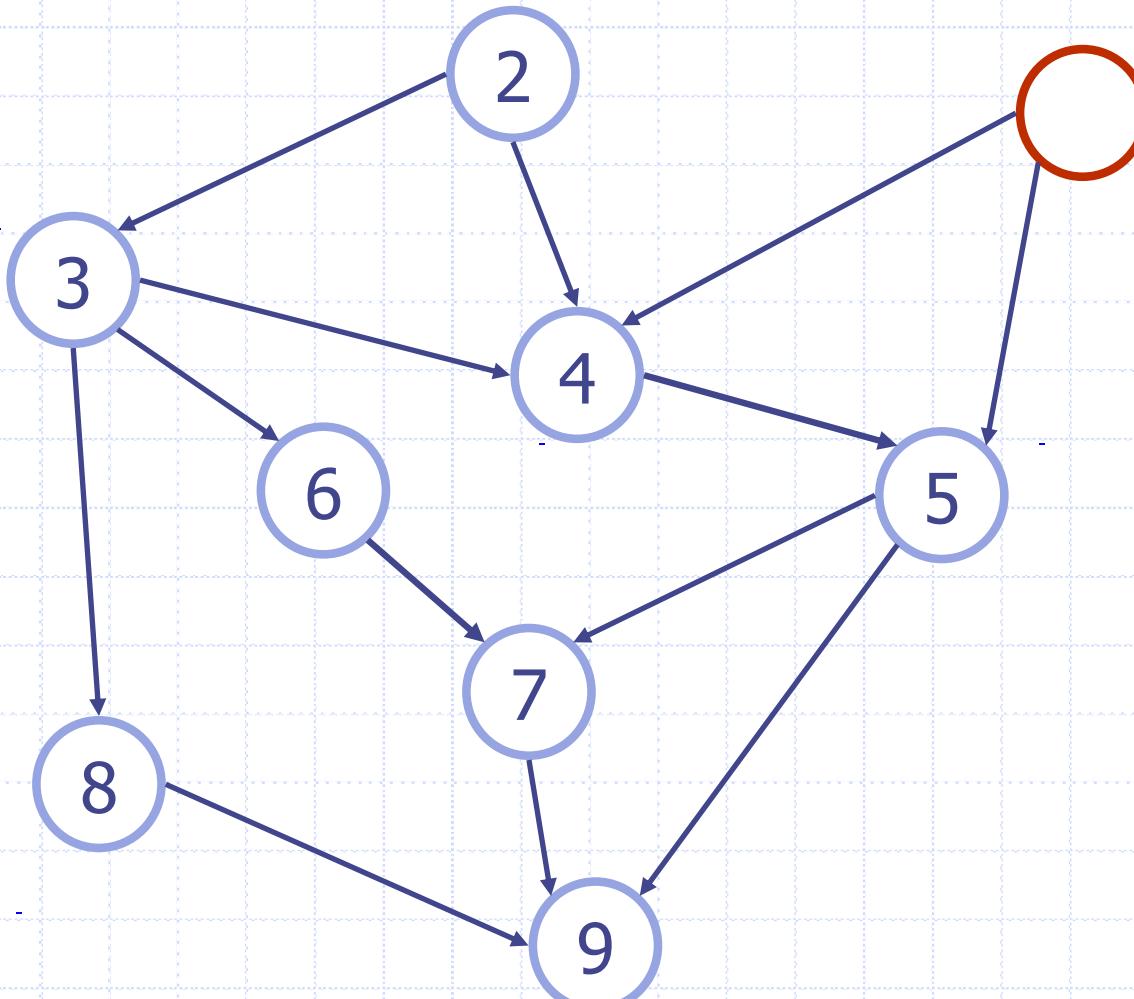
# Ejemplo: Clasificación Topológica



# Ejemplo: Clasificación Topológica



# Ejemplo: Clasificación Topológica



# Ejemplo: Clasificación Topológica

